

# Documentación proyecto programación concurrente y distribuida

1<sup>st</sup> Sebastián Gálvez Yepes

dept. Sistemas e informática

Universidad de Caldas)

Manizales , Caldas

sebastian.1701821561@ucaldas.edu.co

2<sup>nd</sup> Yeraldin Arboleda Quintero

dept. Sistemas e informática

Universidad de Caldas

Manizales , Caldas

yeraldin.1701811368@ucaldas.edu.co

3<sup>rd</sup> Valeria Ávila Nieto

dept. Sistemas e informática

Universidad de Caldas

Manizales , Caldas

valeria.1701520831@ucaldas.edu.co

**Index Terms**—Multiprocessing, pyCuda ,mpi4py , dotplot

## I. INTRODUCCIÓN

El análisis comparativo de secuencias es una técnica fundamental en bioinformática que permite identificar similitudes y diferencias entre secuencias de ADN o proteínas. Una de las herramientas más utilizadas para este propósito es el dotplot, una representación gráfica que facilita la visualización de regiones homólogas entre dos secuencias biológicas. A medida que la cantidad de datos genómicos disponibles crece exponencialmente, la necesidad de métodos eficientes y escalables para generar dotplots se vuelve crucial.

En este proyecto, nos proponemos implementar y analizar el rendimiento de varias aproximaciones para la generación de dotplots. En particular, compararemos las siguientes implementaciones:

- Secuencial: Un enfoque tradicional donde el cálculo del dotplot se realiza de manera lineal.
- Multiprocessing: Utilización de la biblioteca multiprocessing de Python para distribuir el trabajo entre múltiples núcleos de CPU.
- MPI (Message Passing Interface): Implementación con mpi4py que permite la paralelización del cálculo en un entorno distribuido, comúnmente utilizado en clusters de computación.
- GPU Computing: Utilización de pyCuda para aprovechar la capacidad de procesamiento paralelo masivo de las tarjetas gráficas.

Además, se implementará una función paralela para el filtrado de la imagen generada, optimizada para detectar líneas diagonales que representan regiones de alta similitud entre las secuencias.

## II. METODOLOGÍA

Con este proyecto buscamos implementar y analizar el rendimiento de tres métodos que realizan diagramas de puntos comúnmente utilizados en bioinformática para comparar secuencias de ADN o proteínas, en nuestro caso serán 2 secuencias bacterianas, Salmonella y E. coli.

### A. Código Secuencial

El código secuencial lee dos cadenas desde archivos FASTA (.fna), además implementa la generación del dotplot utilizando una ventana deslizante para comparar subsecciones de las secuencias de entrada. Además, monitorea el uso de memoria y guarda imágenes intermedias del dotplot para evitar la pérdida de progreso en caso de que la ejecución sea interrumpida.

### B. Código Multiprocessing

El procesamiento paralelo con multiprocessing en Python permite un análisis eficiente y escalable de secuencias de ADN, optimizando tanto el tiempo como los recursos computacionales. Esta técnica es particularmente útil para tareas computacionalmente intensivas, como la generación de dotplots para grandes secuencias de ADN. El propósito del código es generar un dotplot utilizando paralelización para mejorar el rendimiento. Esto se logra mediante

### C. Código MPI

El procesamiento paralelo con MPI4Py en Python permite un análisis eficiente y escalable de secuencias de ADN, optimizando tanto el tiempo como los recursos computacionales. Esta técnica es particularmente útil para tareas computacionalmente intensivas, como la generación de dotplots para grandes secuencias de ADN. El propósito del código es generar un dotplot utilizando paralelización para mejorar el rendimiento, distribuyendo el trabajo entre múltiples procesos.

Aplicación al Proyecto:

Argumentación y Explicación del Uso de MPI4PY Ventajas del Paralelismo con MPI4PY:

- Escalabilidad: Permite utilizar múltiples nodos en una red, distribuyendo eficientemente el trabajo.
- Rendimiento: La paralelización reduce significativamente los tiempos de ejecución.
- Flexibilidad: Permite gestionar la comunicación entre procesos, optimizando la distribución de tareas.
- Dotplot: Cada proceso calcula una submatriz del dotplot, dividiendo el trabajo de manera eficiente.
- Filtrado de Imágenes: La capacidad de procesar grandes imágenes en paralelo mejora el rendimiento y permite manejar conjuntos de datos más grandes.

### Métricas de Rendimiento:

- **Tiempos de Ejecución:** Se mide el tiempo total y el tiempo de la porción paralelizable para evaluar la eficiencia del paralelismo.
- **Aceleración y Eficiencia:** Se compara la versión secuencial con la paralela para determinar la mejora en el rendimiento.
- **Escalabilidad:** Se evalúa cómo el rendimiento mejora al aumentar el número de procesos.

El uso de MPI4PY en el proyecto permite implementar una versión paralela del dotplot y la función de filtrado de imagen, optimizando significativamente el rendimiento. Al dividir la tarea entre múltiples procesos, se pueden manejar secuencias y imágenes de gran tamaño de manera eficiente, proporcionando una base sólida para un análisis de rendimiento detallado y una comparación robusta con las implementaciones secuenciales y otras técnicas de paralelización.

### D. Código PYCUDA

El procesamiento paralelo con PyCUDA en Python permite un análisis eficiente y escalable de secuencias de ADN, optimizando tanto el tiempo como los recursos computacionales mediante el uso de la potencia de las GPU. Esta técnica es particularmente útil para tareas computacionalmente intensivas, como la generación de dotplots para grandes secuencias de ADN.

El propósito del código es generar un dotplot utilizando la paralelización en GPU para mejorar el rendimiento. Esto se logra mediante PyCUDA, que facilita la ejecución de operaciones paralelas masivas en la tarjeta gráfica. Al aprovechar la arquitectura altamente paralela de las GPU, PyCUDA permite realizar cálculos simultáneos en miles de núcleos, reduciendo significativamente el tiempo de procesamiento en comparación con las implementaciones secuenciales o incluso paralelas en CPU.

Mediante el uso de PyCUDA, se pueden dividir las tareas computacionales intensivas en sub-tareas más pequeñas que se ejecutan en paralelo, aumentando la velocidad de procesamiento y permitiendo el análisis de secuencias de ADN de mayor tamaño y complejidad. Esta metodología no solo mejora la eficiencia, sino que también optimiza el uso de los recursos computacionales, proporcionando una solución escalable y robusta para el análisis de datos biológicos.

### E. Código Filtrado de imagen

Para detectar líneas diagonales en la imagen generada del dotplot, se utiliza un algoritmo de procesamiento de imágenes que se ejecuta en paralelo para mejorar el rendimiento. La detección de diagonales es crucial, ya que estas líneas indican regiones de similitud entre las secuencias comparadas, lo cual es esencial en el análisis de secuencias en bioinformática. El procesamiento paralelo se utiliza para mejorar el rendimiento del filtrado de imagen, permitiendo la detección eficiente de líneas diagonales en grandes imágenes de dotplot.

## III. RESULTADOS

Para los dotplots cada punto indica una región de similitud entre las dos secuencias comparadas. Un punto en la posición (x, y) sugiere que el nucleótido en la posición x en la secuencia de E. coli es similar al nucleótido en la posición y en la secuencia de Salmonella.

### A. Secuencial

La implementación secuencial muestra una ejecución lineal, donde cada parte del procesamiento es manejada de manera secuencial. Esto puede resultar en tiempos de ejecución más largos, especialmente para grandes conjuntos de datos. En comparación con la implementación con multiprocessing, el tiempo de ejecución es mayor, esto muestra la necesidad de optimización para manejar grandes volúmenes de datos de manera más eficiente. Las diagonales se pueden observar como áreas de mayor densidad de puntos que se extienden de manera diagonal a través del gráfico.

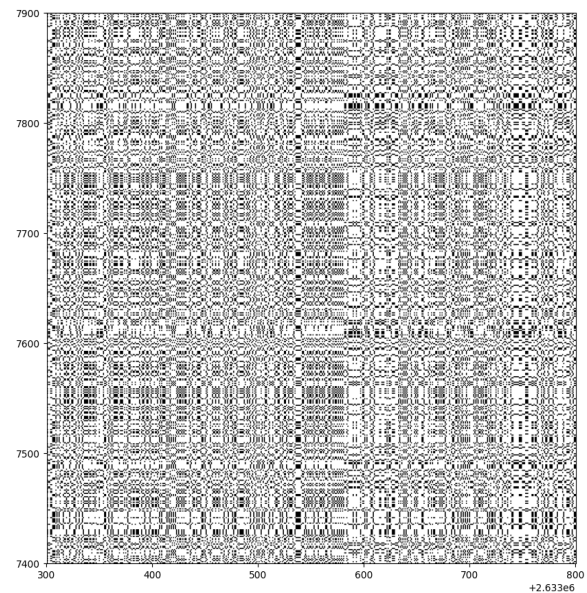


Fig. 1. Dotplot secuencial

### B. Multiprocessing

Las diagonales en el dotplot indican regiones de similitud continua entre las secuencias. Las diagonales largas y continuas sugieren segmentos de ADN que son similares en ambas especies, lo que puede ser indicativo de regiones conservadas evolutivamente. En el dotplot de la imagen 2, la detección de diagonales no es claramente visible debido a la densidad de los puntos. Esto puede ser resultado de la alta similitud global entre las secuencias, mostrando muchas pequeñas similitudes dispersas en lugar de pocas y largas diagonales.

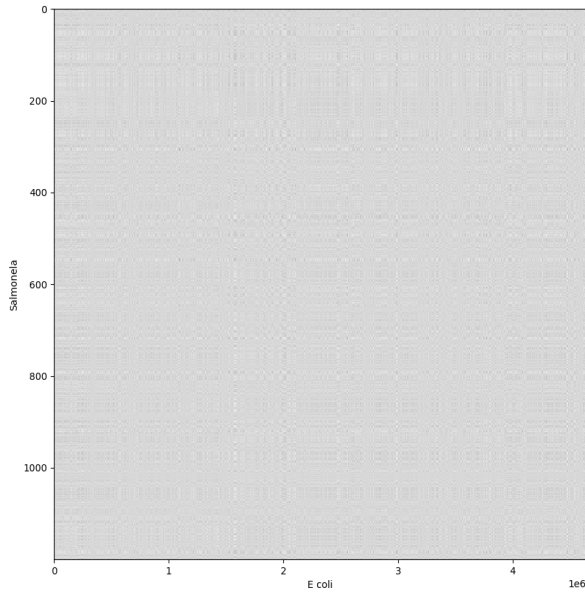


Fig. 2. Dotplot Multiprocessing

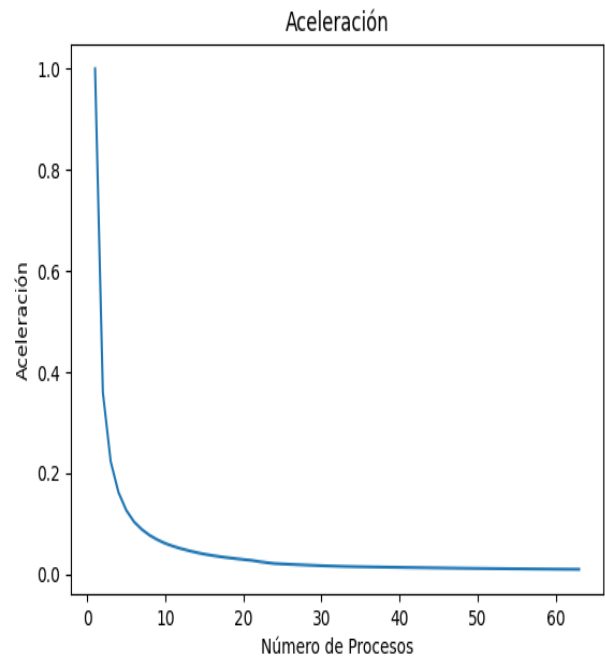


Fig. 3. Multiprocessing aceleración

- En la imagen 3 La aceleración inicial es alta con un número bajo de procesos, pero decrece rápidamente al incrementar el número de procesos. Esto sugiere que hay una ganancia significativa en rendimiento cuando se usan unos pocos procesos, pero esta ganancia se estabiliza y eventualmente disminuye con un mayor número de procesos. Este comportamiento puede estar relacionado con la sobrecarga de gestión de procesos adicionales y la naturaleza del problema de dotplot que puede no escalar linealmente con la paralelización.
- La eficiencia se define como la aceleración dividida por el número de procesos, y refleja cuán efectivamente los recursos de procesamiento están siendo utilizados. La grafica 4 muestra que la eficiencia disminuye drásticamente a medida que aumenta el número de procesos. Esto es esperado, ya que, aunque se agregan más procesos, la sobrecarga de coordinación y la comunicación entre procesos crece, disminuyendo la eficiencia general del sistema. Este comportamiento subraya la importancia de encontrar un equilibrio óptimo en el número de procesos utilizados.
- La escalabilidad evalúa cómo se desempeña el sistema al incrementar el número de procesos. En la Figura 5, se observa que la escalabilidad presenta una caída significativa después de los primeros procesos. Al igual que con la aceleración y eficiencia, este patrón indica que hay un límite en el número de procesos que se pueden agregar antes de que la sobrecarga de administración y comunicación empiece a impactar negativamente el

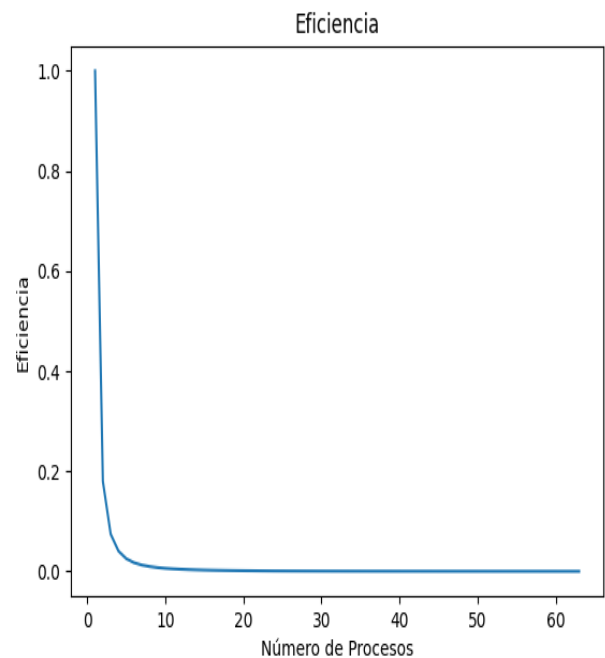


Fig. 4. Multiprocessing eficiencia

rendimiento.

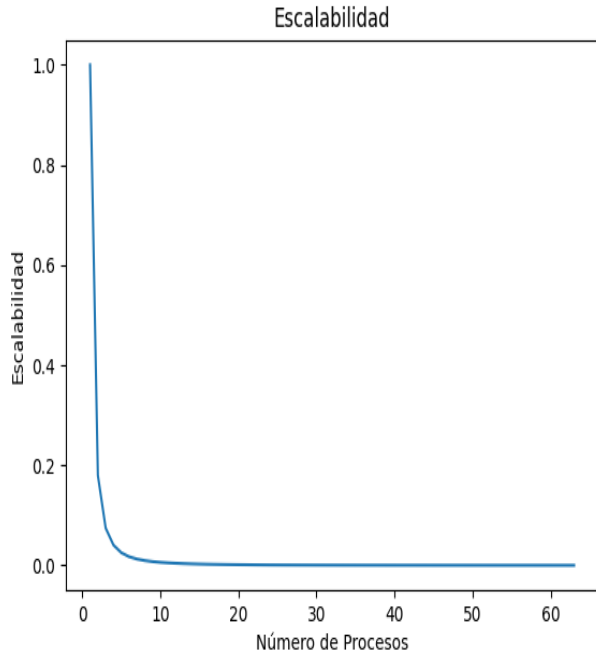


Fig. 5. Multiprocessing escalabilidad

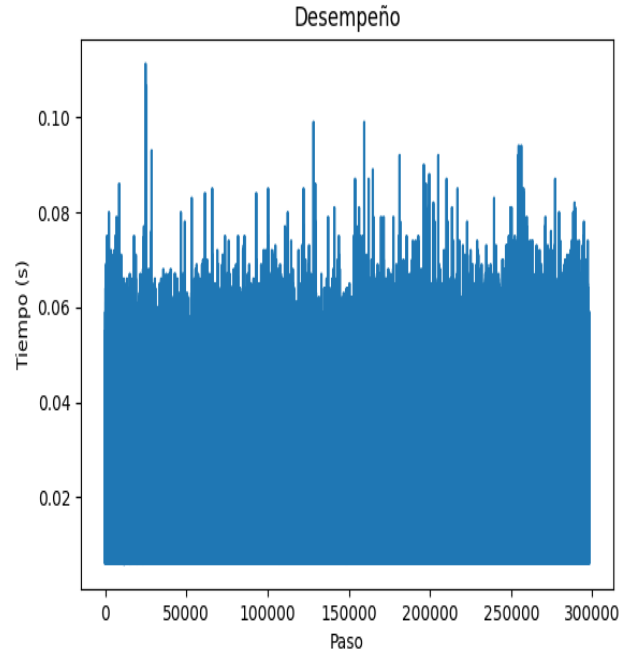


Fig. 6. Desempeño MPI4PY

### C. MPI4PY.

### D. PYCuda

el dotplot de la imagen 9, se pueden observar los puntos de de similitud en la secuencia de ADN

- En la Figura 10, Se observa que la aceleración en el código de PyCUDA aumenta a medida que se incrementa el número de hilos, en comparación con el código secuencial. Esto se debe a la capacidad de las GPU para manejar múltiples hilos simultáneamente, aprovechando el paralelismo masivo que ofrecen. En el código secuencial, las tareas se ejecutan una tras otra, lo que limita la velocidad de procesamiento. En cambio, PyCUDA permite distribuir las tareas entre muchos hilos, reduciendo significativamente el tiempo de ejecución. Además, el rendimiento de PyCUDA mejora notablemente con un aumento en el número de hilos hasta alcanzar un punto de saturación, donde la adición de más hilos ya no proporciona mejoras significativas debido a las limitaciones de hardware y a la sobrecarga de gestión de hilos. Por lo tanto, es crucial encontrar un equilibrio óptimo en el número de hilos para maximizar la eficiencia del procesamiento paralelo.
- En la Figura 11, Se observa que la eficiencia en el código de PyCUDA aumenta considerablemente en comparación con el código secuencial, a medida que se incrementa el número de hilos. Sin embargo, esta eficiencia puede disminuir en relación con el número excesivo de hilos.

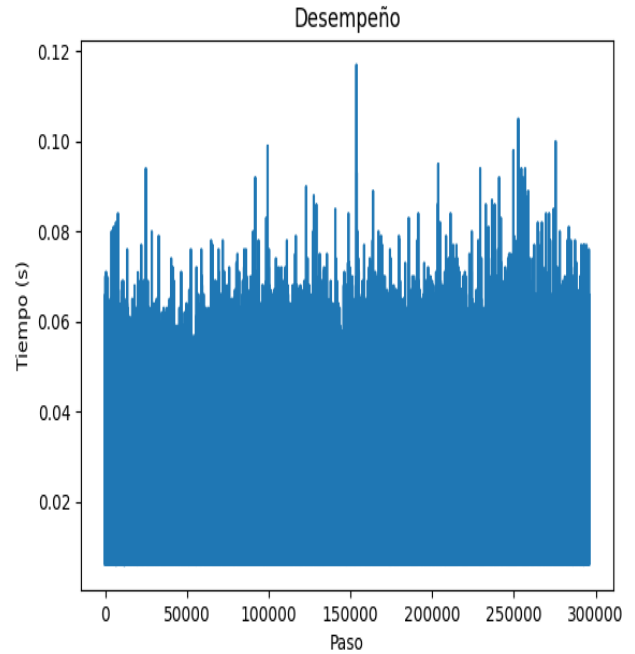


Fig. 7. Desempeño 2 MPI4PY

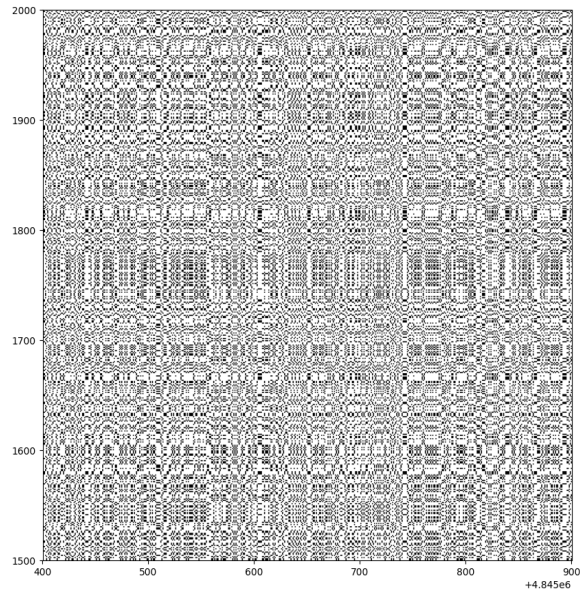


Fig. 8. Dotplot 1 MPI4PY

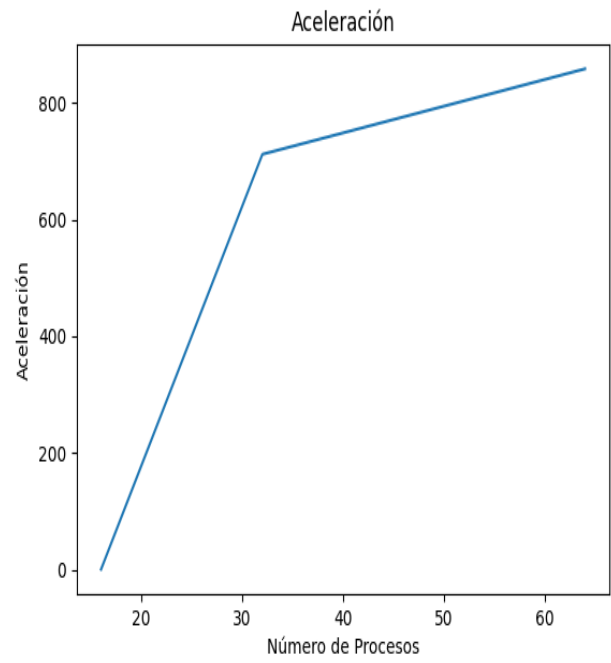


Fig. 10. aceleracion Pycuda

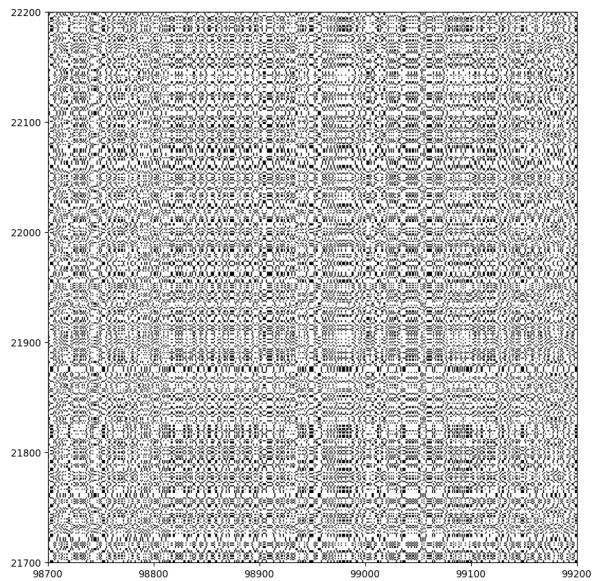


Fig. 9. Dotplot Pycuda

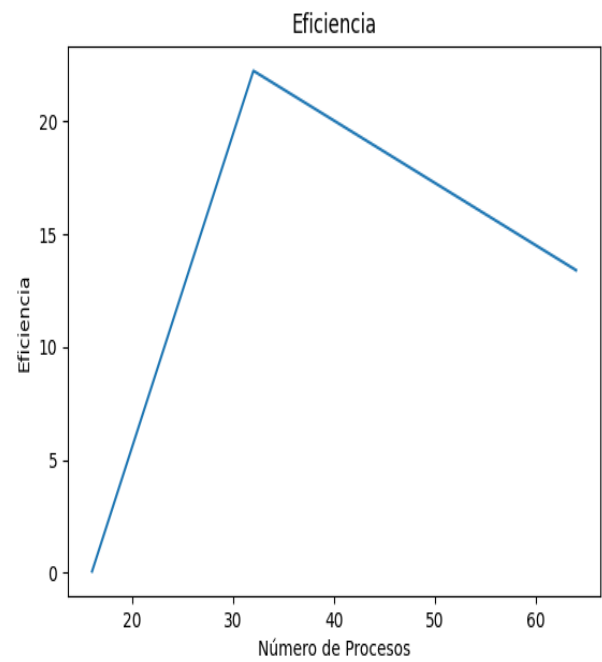


Fig. 11. Eficiencia Pycuda



La eficiencia mejorada con PyCUDA se debe a la capacidad de las GPU para manejar múltiples hilos simultáneamente, aprovechando el paralelismo masivo que ofrecen. Esto permite que las tareas se ejecuten de manera más rápida y eficiente en comparación con la ejecución secuencial, donde las tareas se realizan una tras otra, limitando el rendimiento general.

Por otro lado, aunque un aumento moderado en el número de hilos puede seguir mejorando la eficiencia, superar cierto umbral puede llevar a una disminución en la eficiencia. Esto se debe a la sobrecarga de gestión de hilos y a las limitaciones de hardware que pueden reducir el rendimiento general del sistema.

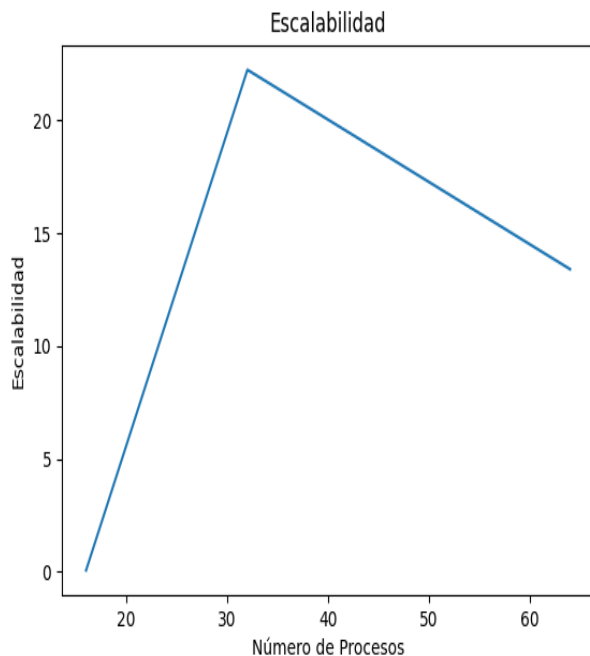


Fig. 12. escalabilidad Pycuda

- En la Figura 12, Se observa que la escalabilidad en el código de PyCUDA mejora significativamente en comparación con el código secuencial a medida que se incrementa el número de hilos. Sin embargo, esta escalabilidad puede verse comprometida si se utilizan un número excesivo de hilos.

La mejora en la escalabilidad con PyCUDA se debe a la capacidad de las GPU para manejar múltiples hilos simultáneamente, aprovechando el paralelismo masivo que ofrecen. Esto permite que las tareas se ejecuten de manera más rápida y eficiente en comparación con la ejecución secuencial, donde las tareas se realizan una tras otra, limitando la capacidad de escalar el rendimiento.

Por otro lado, aunque un aumento moderado en el número de hilos puede seguir mejorando la escalabilidad, superar cierto umbral puede llevar a una disminución en la escalabilidad. Esto se debe a la sobrecarga de gestión de

hilos y a las limitaciones de hardware que pueden afectar negativamente la capacidad de escalar eficientemente.

## IV. DISCUSION

### A. Código Secuencial

**Ventajas:** Simplicidad de implementación. Fácil depuración y mantenimiento.

**Desventajas:** Ineficiente para secuencias largas debido a la falta de paralelización. Tiempo de ejecución prolongado ya que el procesamiento se realiza en un solo núcleo de CPU.

**Conclusión:** Adecuado solo para secuencias pequeñas y casos donde la simplicidad es más importante que el rendimiento.

### B. Código Multiprocessing

**Ventajas:** Utiliza múltiples núcleos de CPU para dividir el trabajo, reduciendo significativamente el tiempo de ejecución. Relativamente fácil de implementar y depurar en comparación con MPI o CUDA.

**Desventajas:** Limitado por el número de núcleos de CPU disponibles en la máquina. La sobrecarga de la creación y gestión de procesos puede afectar el rendimiento para tareas muy pequeñas.

**Conclusión:** Buena opción para mejorar el rendimiento en máquinas con múltiples núcleos de CPU, especialmente para secuencias de tamaño moderado.

### C. Código MPI

**Ventajas:** Permite la paralelización en múltiples nodos de un clúster, no solo en múltiples núcleos de una sola máquina. Escalable para tareas muy grandes y distribuidas.

**Desventajas:** Requiere un entorno de clúster y la configuración de MPI, lo que puede ser complejo. La comunicación entre nodos puede añadir sobrecarga y latencia.

**Conclusión:** Ideal para entornos de clúster donde se necesita procesar secuencias extremadamente largas o grandes volúmenes de datos distribuidos.

### D. Código Pycuda

**Ventajas:** Utiliza la GPU para paralelizar el procesamiento, lo que puede ofrecer una aceleración significativa para tareas masivamente paralelas. Excelente rendimiento para operaciones vectoriales y matriciales comunes en bioinformática.

**Desventajas:** Requiere conocimientos de programación en CUDA y la arquitectura de la GPU. Dependiente del hardware GPU disponible, y la implementación y depuración pueden ser más complejas.

**Conclusión:** La opción más eficiente en términos de tiempo de ejecución para tareas adecuadas para la aceleración por GPU, especialmente en hardware con potentes capacidades de cómputo de GPU.

## V. CONCLUSIONES

Para secuencias pequeñas nn código secuencial puede ser suficiente debido a su simplicidad.

Para secuencias de tamaño moderado y en máquinas con múltiples núcleos de CPU el multiprocessing es una excelente opción que mejora el rendimiento sin una complejidad excesiva.

Para secuencias extremadamente largas y entornos de clúster MPI4PY es adecuado, permitiendo una paralelización distribuida y escalable.

Para tareas que pueden beneficiarse de la aceleración de la GPU, PyCUDA es la mejor opción, ofreciendo el mejor rendimiento en hardware compatible.

PyCUDA tiende a ser la opción más eficiente en términos de rendimiento puro debido a la capacidad de la GPU para manejar tareas masivamente paralelas. Sin embargo, la mejor elección depende de los requisitos específicos del proyecto, la disponibilidad de hardware y los conocimientos técnicos del equipo de desarrollo.

## REFERENCES

- [1] “Diferencia entre E Coli y Salmonella / Biología,” La diferencia entre objetos y términos similares., 2024. <https://es.differkinome.com/articles/biology-science-nature/difference-between-e-coli-and-salmonella.html> (acceso Jun. 12, 2024).
- [2] B. Mather, “Multiprocessing in Python,” Ben Mather, Nov. 24, 2018. <https://www.benmather.info/post/2018-11-24-multiprocessing-in-python/> (acceso Jun. 12, 2024).
- [3] R. Sharma, N. Gupta, V. Narang, and A. Mittal, “Parallel implementation of DNA sequences matching algorithms using PWM on GPU architecture,” International journal of bioinformatics research and applications, vol. 7, no. 2, pp. 202–202, Jan. 2011, doi: <https://doi.org/10.1504/ijbra.2011.040097>.